Bayan: A Text Database Management System which Supports a Full Representation of the Arabic Language

Ali Morfeq
Roger King

Department of Computer Science
University of Colorado
Boulder, Colorado 80309

Aqil Asmi

Center for Hadith Analysis
Boulder, Colorado 80303

ABSTRACT

It is difficult to use existing computer systems, based on the Roman alphabet, for languages that are quite different from those that use the Roman alphabet. Many projects have attempted to use conventional systems for Arabic, but because of Arabic's many differences with English, these projects have met with limited success. In the Bayan project, the approach has been different. Instead of simply trying to adopt an environment to Arabic, the properties of the Arabic language were the starting point and everything was designed to meet the needs of Arabic, thus avoiding the shortcomings of other Arabic projects. A new representation of the Arabic characters was designed; a complete Arabic keyboard layout was created; and a window-based Arabic user interface was also designed. A graphical based windowing system was used to properly draw Arabic characters. Bayan is based on an object-oriented approach which helps the extensibility of the system for future use. Furthermore, linguistic algorithms (for word generation and morphological decomposition of words) were designed, leading to the formalization of rules of Arabic language writing and sentence construction.

1. Introduction

Arabic language processing has been hampered because, for the most part, people have simply attempted to adopt English-based computer systems for Arabic such as [TAYL86]. This is very difficult and, to some extent, futile because the nature of the Arabic language is much different than that of English. Arabic, for example, is written from right to left; thus, the Arabic sentence structure and grammar are very different from that in English. Therefore, existing systems ("adopted English systems") cannot adequately process Arabic language data. In Bayan, on the other hand, the Arabic properties are served first. This new environment will try to examine the problems in working with the Arabic language by using Arabic text databases as an application of that environment.

This research will address those problems that have led to the shortcomings in the current Arabic programs. In the next section, an overview of the properties of Arabic will be given so the reader may appreciate some of the possible problems in dealing with Arabic using an English-based system. Some aspects of Arabic word processing requirements can be found in [BECK87] [AHME86].
2. The Arabic Language

Arabic is the official language of 22 countries stretching from eastern Asia to northwest Africa. Furthermore, there are other languages which use a modified Arabic alphabet (such as Persian and Urdu). The importance of Arabic is perhaps obvious to the reader.

2.1. Properties of Arabic script

The following subsections describe the general properties of the Arabic script, including characters, digits and writing methods.

Arabic characters

There are 28 characters in the Arabic alphabet. The sounds of 12 of them do not map directly to sounds that are part of the English language. The following are some important properties of the alphabet.

- Arabic may only be written in cursive script.
- Arabic is written from right to left.
- In writing, each character is connected to the preceding or following characters, except six characters, called "stubborn characters". These six stubborn characters are { و د ذ ر ز و }
- Each character, in general, takes on a different shape depending on whether it appears in the beginning, middle or end of a word or if it is standing by itself (for example, after a "stubborn character" at the end of a word).
- The Arabic alphabet contains no vowels as such. Instead, diacritical marks, called "tashkeel", are used to represent vowel sounds. They are written above or below the consonants. There are 13 "tashkeel". Four "tashkeel" are basic and nine are derived by combining one or more "tashkeel". The "Tashkeel" are essential to understanding the sentence properly; without the tashkeel, a sentence may easily be misunderstood. This is because the placement of the "tashkeel" is determined by the rules of grammar; that is, the "tashkeel" tells the reader if the word is a verb, subject, adjective and so on.
- Each character in Arabic could be pronounced in 13 different ways, depending on the "tashkeel" written with it.

Arabic numerals

Although the numerals of the English language are referred to as Arabic numerals, they differ completely from the numerals used throughout the Arabic world.

There are ten digits in Arabic. These digits are {١ ٢ ٣ ٤ ٥ ٦ ٧ ٨ ٩ ٠ }. The numbers are written and read from left to right (not right to left, like Arabic words). This implies, obviously, that the least significant digit in a number is the right most digit.

2.2. Existing standards

There have been many attempts to simply modify English computer systems to accommodate Arabic. Such attempts have ranged from modifying input and output (I/O) device drivers to using graphics to draw the Arabic script within application programs. All of these projects have failed in their attempt to represent all of the aspects of the Arabic language. For example, many of them do not have any means of representing the "tashkeel".

One of the proposed standards for representing Arabic characters in computers may be found in [ALSA86]. That work contains a proposed representation of a 7-bit Arabic character set (without Tashkeel) and an 8-bit set for both Arabic (without Tashkeel) and English. Still these sets are not able to meet all the requirements of satisfactorily representing the Arabic language. The source of these shortcomings, again, is because these character sets were intended to work with English machines.

In general, the proposed standards lack the following:

- The ability to represent all of the Arabic "tashkeel", especially the ones that are used as a combination of "tashkeel" (e.g., "".).
In general, they represent "tashkeel" as characters added to a word. Two words may have the same characters but differ in meaning because of their different "tashkeel". When considering "tashkeel" as characters added to a word, this difference cannot be taken into consideration.

They arrange some of the characters with special markings as unique characters while they are simply the same character with different signs. For example, (ٍ ٌ ّ) are all the same character, called "hamza". The system must consider all of them as one character and not as different characters.

Finally, these standards are not accepted by all Arab countries and, therefore, until now there is no uniformity on this matter.

Most of the proposed standards are simply derivations from [ALSA86]. Some simply change the ordering of the characters while others remove or add some extra characters. There are some application programs which tried to solve some of the problems related to the "tashkeel" by creating a character set which included each of the "tashkeel". However, since there are only 127 character entries in the 7-bit extended character sets, they were forced to ignore some of the "tashkeel" in their implementation.

3. The Bayan environment

The Bayan environment attempts to solve the problems described above by attempting to fully represent Arabic script. In particular, representing the different forms of "tashkeel" is given high priority. Bayan also includes a database of Arabic words, roots and derivations rules. This environment could be a base for future Arabic natural language processing research.

The Arabic character set

A 16-bit coded method is to be used in the Bayan system. Arabic characters are encoded in such a way that each character is represented by one and only one code. That is, the "tashkeel" do not affect the value of the character. Instead, the "tashkeel" are treated as a description or a property of the character. They can be treated in the same way underlining, highlighting and other screen display information are treated. A character will be represented by a byte while attributes (such as underlining, highlighting and so on) are represented by another byte. It was decided to represent Arabic characters as in Figure 1. In this figure, the right hand side byte is used to represent the characters while the left hand side byte is used to describe the properties of the character (i.e. the "tashkeel").

![Figure 1: A proposed representation of an Arabic Character](image)

It may be possible to make the code-byte values a modification of the proposed standard found in [ALSA86] in order to allow some compatibility with the proposed standard. However, the character forms that represent the "tashkeel" are removed as, it is argued here, it is more efficient not to include the "tashkeel" in the character set: rather, they should be considered as attributes to each character. Furthermore, it is also proposed here, that each character should be stored in one code only.

Output drivers must be able to determine the proper shape of each character (depending on where it occurs in the word) along with the correct positioning of the "tashkeel". This is done by building a contextual analyzer which knows how Arabic script is written.

An Arabic keyboard

Since there are only 28 characters in the Arabic alphabet, it is possible to use an English keyboard and simply modify it to resemble the Arabic typewriter keyboard. The keyboard Bayan is shown in Figure 2. Note that the "tashkeel" have keys on the Bayan keyboard, but Arabic typewriters typically do not have "tashkeel" (the tashkeel are added by hand). Device drivers should be able to map the scanning code from the keyboard to the appropriate Arabic code and attribute. This keyboard layout is a collection of different proposed layouts and it overlaps with many of them.
The default input method is to enter the Arabic character followed by its “tashkeel”. However, an editor has been added in order to allow the user to first type characters without entering the “tashkeel” and to go back and add the proper “tashkeel” later if so desired. This feature was added because some users prefer to write a word first and then go back and add any desired “tashkeel”.

4. Contextual analysis

In Bayan, a contextual analyzer was built. The role of this analyzer is to determine the shape of the Arabic character according to its position in a word. The analyzer takes into consideration whether the character appears at the beginning, middle or end of the word. It will represent the character properly and also place the “tashkeel” in their proper place.

The block diagram of the contextual analyzer is shown in Figure 3. It uses four fonts for the output of Arabic characters. There is a separate font for each possible position or shape of a character. These fonts are: “stand alone font”, “start of word font”, “connect at middle font”, and “connect at end font”. The analyzer will take the current and previous character codes and their shapes as its input and will return the correct shapes of the previous and current character. This means that the calling procedure to the contextual analyzer should check if the previous character changes shape after the return from the contextual analyzer call.

The algorithm to determine the shape of the character at the current position is shown in Algorithm 1. It uses the following structure and identifiers.
Data structures and constants.
identifier:
START_POSITION, MID_CONNECT, END_CONNECT, and STAND_ALONE;

Arabic_Char Structure of {
Code : Byte; /*Arabic character code*/
Tashkeel : Byte; /*Tashkeel code*/
Shape : int; /*character shape number*/
}

*******************************************************************************
contextual analysis for the Arabic language
written in C like language. Stuff between /* */ are comments.
*/
Procedure ( current_chr, Arabic_chr, prev_chr, Arabic_char) {
  current_chr.shape = MID_CONNECT; /* default shape */
  if( CAN_NOT_CONNECT(prev_chr.code) ) /* check the prev chr */
    current_chr.shape = START_POSITION;
  exit();
  } /* if can_not_connect */

if( NOT_LETTER(current_chr.code)) /*check current character*/
  Current_chr_shape = START_POSITION; /* current will be start_shape*/
  do case (prev_chr.shape)
    case START_POSITION: prev_chr.shape = STAND_ALONE;
    break;
    case MID_CONNECT: prev_chr.shape = END_CONNECT;
    break;
    case END_CONNECT:
    case STAND_ALONE: break;
    default: /* ERROR("SHAPE code"); */
  /*do case*/
} /* if_non_letter */
else {
  do case (prev_chr.shape)
    case START_POSITION:
    case MID_CONNECT: current_chr.shape = MID_CONNECT;
    break;
    case END_CONNECT:
    case STAND_ALONE: current_chr.shape = MID_CONNECT;
    current_chr_shape = MID_CONNECT;
    break;
    default: /* ERROR("ERROR in shape code"); */
    /* do case prev_chr.shape */
  /* else */
  return();
} /* end of procedure */

The identifiers above are used to represent the different fonts used in our system. For example, START_POSITION is the font that contains the shape of all characters when they are at the starting position of a word.

The contextual algorithm will take the current and previous characters as inputs. The contextual analysis algorithm will examine the previous and current characters. If the previous character can not connect to the current character (e.g. space, comma, a stubborn character...), then it will make the current character's shape to be "START_POSITION SHAPE" and it will exit. Next, the algorithm will check if the current character is a space, a punctuation mark, or a digit; in which case, it will set the shape of the previous character to be the proper ending shape depending on the shape of its previous character(i.e. the second to the last character). If both of the above tests fail, the algorithm will change the current character to connect to the previous character depending on the shape
of its previous character.

In the following table, we show an example of how contextual analysis determines the shape of the Arabic character displayed. The first column shows the current input character. The second column shows what is being displayed as a result of the input so far. The third column contains comments of what has been done. The black rectangle is the cursor.

<table>
<thead>
<tr>
<th>Input character</th>
<th>Output on screen</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>س</td>
<td>س</td>
<td>start of word shape.</td>
</tr>
<tr>
<td>ل</td>
<td>مل</td>
<td>connect the second to the first letter.</td>
</tr>
<tr>
<td>ط</td>
<td>ط</td>
<td>a ligature is a must, so go back and join both into one new char.</td>
</tr>
<tr>
<td>ﷺ</td>
<td>ﷺ</td>
<td>start of word shape because the previous can not connect to next.</td>
</tr>
<tr>
<td>سلام</td>
<td>ﷺ</td>
<td>space cause the last letter to have ending shape.</td>
</tr>
</tbody>
</table>

5. Bayan's user interface

Bayan’s user interface was built in a SUN workstation environment. Machine dependency was kept to a minimum. The existing interface will allow the use of Arabic text in windows; it will also perform all proper input and output operations on that window. Furthermore, Arabic text may be edited in these windows.

Figure 4
Arabic window in Bayan.

17.
Figure 4 shows an Arabic window which displays Arabic text information. Note that the text has displayed the proper “tashkeel” for every word of text. The figure shows the main options allowed by the system. Currently the Text_unit option is activated. The Text_unit object called “Bukh7.new” is displayed. The other options are document, query, learn and exit options. The document option, for example, allows the user to manipulate and create documents.

6. The Arabic word manager

This manager is responsible for deriving words from their roots and reporting the resulting word to the requesting procedure. This manager must know the rules for word derivations in Arabic as well as have the ability to learn new rules. This feature is built into the word manager because all rules of derivation that can apply to a word may not be known to the user when a word is added to the system. This feature will allow the user to add additional rules later if needed. Figure 5 shows the major components of the word manager.

![Diagram of word manager components]

This manager allows a database administrator (DBA) to enter what is known as the teaching mode. In the teaching mode, the learning manager is called and then the DBA can add new words to the vocabulary of the system, add new derivation rules and specify which words it applies to. In the teaching mode, the DBA can modify existing derivation rules for the roots of an Arabic word.

7. The learning manager

This part of the word manager can learn new words. It takes information about a word and adds it to the object manager. It is important that this manager only be used by someone proficient in Arabic as incorrect information can be harmful to the database since Bayan stores words as objects rather than actual characters of a word.

Derivation rules are defined for each word by use of the learning manager. This is the only way to add a word to the vocabulary of the system. Arabic words are added to the database objects through this option after obtaining the required information to form a complete word object and to check if the word already exists.

The learning manager uses an interactive window-based interface that prompts the user to supply the required information. This process guides the DBA by requesting the needed information before accepting the new word. The DBA is responsible for the correctness of the information in the system.

The learning manager can also accept new derivation rules. Rules can be specified by the DBA using the proper format. Rules can be added to the knowledgebase of the learning manager and are used by the word generation engine.

18.
8. The word generation engine

The word generation engine interprets derivation rules and executes commands encoded in the derivation rules. It is like a small computer executing a limited instruction set. It will take a root as an input and produce the newly generated word (if any) as output. If there is an error, the same root will be returned with a flag indicating the presence of an error. Errors can range from a null root to the wrong number of characters in the root and so on.

8.1. Derivation rules

All the rules are the same for each root class of words. Therefore, they need to be stored only once; afterwards, the word generation manager can generate the desired word from its root. It should be noted that rules are completely different for three character roots, four character roots and five character roots.

In general, a derivation is performed by adding characters and/or changing the "tashkeel" of the word. The encoding of the derivation is as follows:

<digit>
concatenate character number <digit> of the root adding the following "tashkeel". Where each <digit> will be followed by the proper "tashkeel".

<character>
concatenate <character> where character represents an Arabic character with its "tashkeel".

An example of what the derivation rules look like is the following:

Several examples of Arabic derivation rules are given below.

Arabic derivation rules, however, do not only involve the concatenation of characters at the end of a word. Characters may also be added to the middle, beginning or end of a word to derive a new word.

8.2. An example:

Given the following:

Root = نَصَرٌ
Rule = مَكْرُ وَ سَوَانُ
Result = مُنْصِرُوُنُ

The steps for this result is shown in the following table:
<table>
<thead>
<tr>
<th>Step number</th>
<th>Resulting word so far</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>مَنْتَوْرُونْ</td>
<td>the new word starts with the arabic character &quot;Meem&quot;</td>
</tr>
<tr>
<td>2</td>
<td>مَنْتَوْرْ</td>
<td>letter number (1) of the root word to be added.</td>
</tr>
<tr>
<td>3</td>
<td>مَنْتَوْرْ</td>
<td>letter number (2) of the root word to be added.</td>
</tr>
<tr>
<td>4</td>
<td>مَتْحُورُ</td>
<td>letter (waow) to be added.</td>
</tr>
<tr>
<td>5</td>
<td>مَتْحُورُ</td>
<td>letter number (3) of the root word to be added.</td>
</tr>
<tr>
<td>6</td>
<td>مَتْحُورُ</td>
<td>letter (waow) to be added.</td>
</tr>
<tr>
<td>7</td>
<td>مَتْحُورُونْ</td>
<td>letter (noon) to be added.</td>
</tr>
</tbody>
</table>

9. Morphological analysis of Arabic

A morphological analyzer for Arabic words is in the process of being built. It will be able to provide the following information concerning Arabic words: recognition of different parts of a word, the root of a word, and affixes attached to words.

The following information is required for the morphological analysis of Arabic words:

1. The roots of words along with all the derivation rules that can be applied to them;
2. The derived word and root links (which can be performed by a lookup table or by a smart algorithm) which takes a word and returns its root;
3. A list of all possible affixes that can be attached to a word and that are not covered by the derivation rules; and
4. A list of words that are not derived from roots, such as tools (e.g., "from", "to", "on" and so on).

10. Scenario

A user will create a text_unit object using bayan’s editor. Figure 4 shows an editing session of a text_unit object called ‘bukh7.new’. External files can also be imported to Bayan’s editor using the “Load” option. An option can be activated by clicking on the button which represents the command. The updated Text_unit can then be stored in the database using the “Store” option. The user, when done, can ask Bayan to incorporate the new or modified Text_unit object into its final database. Bayan will check the Text_unit object for new words, index the words, or it can activate the learning manager if a word is not known to Bayan.

The user can specify the information which Bayan’s learning manager prompts him/her for. This will include the specification of which class of Arabic words this word belongs to. The user will also verify the derivation method used to derive this word from its root. Additional information may be requested to determine affixes added to the word. If the user made a mistake in the word, he can go back and modify the Text_unit object using the editor, then ask Bayan to continue its operation. Bayan will add this information to its database. The user can query that Text_unit object by words that appear in it.

Documents can be built using the “Document option” which can be selected from the main menu. The user will be required to enter the elements which make up documents. These elements
may include Text_unit objects and/or other documents.

The user can make queries about words within document objects. The result of the query will be a list of the document objects in which the word appears. The user can then specify that he wants to see the Text_unit object level and the Text_unit objects will be displayed. At the Text_unit object level the user can browse or modify these objects. If he modifies a Text_unit object he would need to ask Bayan to update its database as explained above.

Queries can also be made about Arabic words, their roots, or their derivation methods. Queries can ask Bayan to show examples of words derived from the same root, or even words of similar meanings.

11. Conclusion

Bayan is a starting step for more advanced research regarding Arabic. It points to the possibility of more advanced studies about the language, such as natural language processing based on Bayan's environment. For now, though, many persistent problems have been addressed, such as the design of an Arabic character set including the "tashkeel", an Arabic keyboard, word-generation algorithms and Arabic word generation methods from the roots of words. At the present time, an Arabic text database, which utilizes the representation and algorithms designed in Bayan, is being built. In addition, a built-in morphological word analyzer is also under construction.

References:


[HASS87] Mohamed Hassoun, 'Conception et réalisation d'une base de données pour les traitements automaïque de la langue arabe', 10th computer conference in Saudi Arabic, 1987